WebFlow – a visual programming paradigm for Web/Java based coarse grain distributed computing

Dimple Bhatia, Vanco Burzevski, Maja Camuseva,
Geoffrey Fox, Wojtek Furmanski and Girish Premchandran

Northeast Parallel Architectures Center at Syracuse University

dbhatia@npac.syr.edu, vanco@top.syr.edu,maja@top.syr.edu,
gcf@npac.syr.edu, furm@npac.syr.edu, girishp@npac.syr.edu

Presented at Workshop on *Java for Computational Science and Engineering Workshop*, Syracuse University, December 1996.

Summary

We present here the recent work at NPAC aimed at developing WebFlow – a general purpose Web based visual interactive programming environment for coarse grain distributed computing. We follow the 3–tier architecture with the central control and integration WebVM layer in tier-2, interacting with the visual graph editor applets in tier-1 (front-end) and the legacy systems in tier-3. WebVM is given by a mesh of Java Web servers such as Jeeves from JavaSoft or Jigsaw from MIT/W3C. All system control structures are implemented as URL-addressable servlets which enable Web browser-based authoring, monitoring, publication, documentation and software distribution tools for distributed computing. We view WebFlow/WEbVM as a promising programming paradigm and coordination model for the exploding volume of Web/Java software, and we illustrate it in a set of ongoing application development activities.

Contents

1 Introduction

As anticipated in our WebWindows ansatz [WebH-PCC96], current Web systems, fueled by Java, evolve rapidly towards a powerful open infrastructure that will enable world-wide distributed computing. In the current Web/Java expansion phase, we are witnessing a wide variety of new interesting tools and technologies but the overall integration framework is still missing and the software reuse remains difficult. We need a coarser grain encapsulation unit than a Java class to enable user-friendly distributed computing on the Web. In fact, several attempts at specifying such a framework are underway, for example JavaBeans from JavaSoft. However, the Web software industry is currently focused mainly on the front–end support for component based GUI integration, whereas the middleware and back-end layers are still an open research and prototyping area.

At NPAC, we are monitoring the emergent Web technologies pertaining to the domain of world wide scalable distributed computing and we are designing and prototyping a visual graph based dataflow environment, WebFlow, using the mesh of Java Web Servers as a control and coordination middleware, WebVM.

In this document, we review briefly our Web technology evaluation activities in Sections 2 – 4, followed by the presentation of our WebFlow/WebVM prototype (tier-2) in Section 4 which forms the core of this paper. This is followed by the discussion of the WebFlow front-end (tier-1) in Section 6 and some initial back-end (tier-3) activities in Section 7. Finally, we summarize in Sections 8 a set of planned or ongoing application development activities in the areas of command and control, telemedicine, distance education and Internet commerce that will build on top of WebFlow/WebVM infrastructure.

	Habanero	**Jigsaw**	**Infospheres**	**JavaSoft**	**Netscape**
Module	collaboratized applet	Resource	dapplet -> DJINN	Java Bean	LiveWire app, server
Port/Channel	Java socket	any HTTP carrier	portlet -> mailbox	RMI	custom?
Message	Marshalled Event or Action	Pickled Resource	any object bytesteam	Serialized Object	JavaScript
Compute-Web	Star topology	2-node	any topology		
Runtime	Collaboratory server	Java HTTP server	dapplet / DJINN manager?	Jeeves (Java server)	community or enterprise system
User Interface	AWT	Forms	visual authoring?	HotJava	Navigator
Coordination	instantaneous broadcast	client-server	asynchronous multi-server	CORBA	multi-server
Persistency		Resource Store	flat file?	JDBC	LiveWire -> DB
Publication				javadoc	

Table 1: Comparative analysis of strategies and components for Web based distributed computing in selected systems investigated at NPAC.

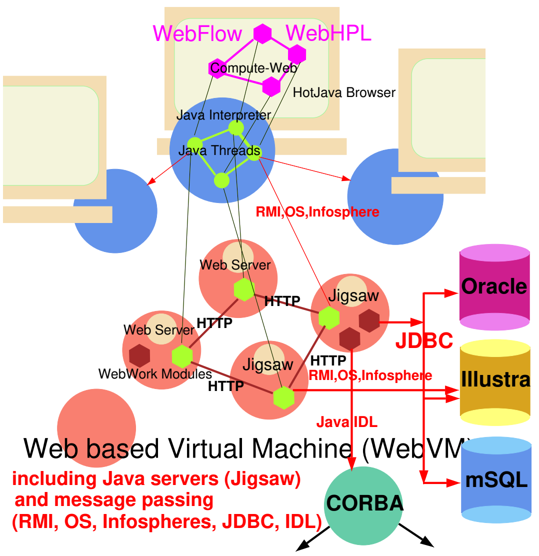


Figure 1: Overview of the WebFlow/WebVM architecture: WebVM is formed in tier-2 as a mesh of Java Web servers, managing WebFlow nets (or *compute-webs*) and interacting with the legacy systems in tier-3 (back-end) and with the visual graph editor applets in tier-1 (front-end).

2 Web/Java Expansion Phase

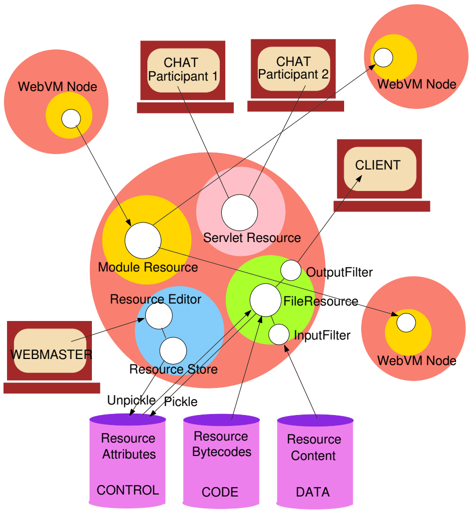
Expressive power of Java attracts developers and we observe an explosion of first generation Java systems on the Internet. Examples include: NCSA Habanero [Haba96] for synchronous collaboratory; dynamic HTTP servers such as Jigsaw [Jigs96] from MIT/W3C or Jeeves from JavaSoft; Marimba's Castanet and Bongo trying to establish a new pure Java based Web-like framework; Caltech Infospheres [ChaRi96], IBM aglets for intelligent agents based computing; and many others.

At NPAC, where we are closely monitoring this 'bleeding edge' of interactive Web, we observe that although these new systems offer attractive capabilities, the current generation Java software is still difficult to customize, repackage or reuse. The reason is that Java class is a too small, too fine grain encapsulation unit and hence reusing a package requires usually detailed understanding of a large number of its tightly interwoven classes.

3 WebFlow/WebVM Concepts

Our goal is to provide a coarser grain packaging model and the associated user-friendly authoring framework for distributed applications on the Web. We believe that we should build on top of the established standards such as HTTP, HTML and Java, and hence we adopt Java Web server as a base runtime and coordination node of any distributed Web system. Dataflow model, already proven effective by previous systems such as AVS, Khoros, CODE [Browne92], HeNCE [Dong94] and others, seems to be a natural coordination framework to extend the current 2–node model in which HTTP/MIME data flows between Web client and server towards multi-server systems.

Hence, we propose a runtime environment given by a mesh of Web Java servers to coordinate distributed computation represented as a set of channel-connected coarse grain Java modules. Modules are thin veneer Java interfaces so that any chunk of Java can be easily modularized and connected to other modules via suitable communication ports, acting as terminals of point-to-point dataflow channels. Modules run asynchronously, are mobile i.e. can be instantiated on any WebVM server, and communicate by exchanging Java objects along their dataflow channels.



Jigsaw Server as WebVM Node

Figure 2: Internal dynamics of the Jigsaw Java Web server by MIT/W3C. All services are structured and managed as Resource objects (similar to Servlet objects in Jeeves). Resources are maintained in a persistent store, editable and downloadable on demand. The figure illustrates a set of standard Jigsaw resources such as File or Editors, and our own experiments with multi-user and/or multi-server extensions such as Chat session or WebFlow module Resources.

Aspects of such emergent architecture can be already found in current systems, analyzed in Table 1. For example: Jigsaw/Jeeves develop the concept or resources/servlets as control encapsulation units; Infospheres develops portlets/mailboxes as terminals for communication channels; Habanero is a multi-server system; and so on.

Integrating Jigsaw with Caltech Infospheres

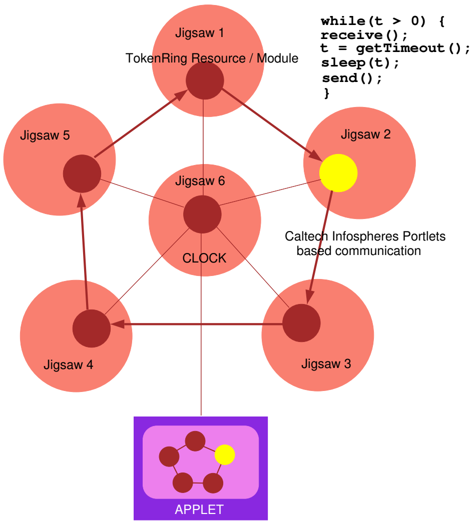


Figure 3: Early integration experiments: Portlet library extracted from Caltech Infospheres is used to form a token ring, connecting a set of Jigsaw Resource nodes. A message packet rotates along the ring with a user-adjustable speed and generates visual feedback in a monitor applet.

4 Early Experiments

We initiated the WebFlow/WebVM design process by experimenting with existing systems. Over the summer/fall '96, we evaluated a suite of new Java systems including Aglets, Habanero, Infospheres, Jeeves, Jigsaw, JSDA, Shaking Hands, and others. One of early decisions we made was that rather then developing custom Java servers from scratch as in Habanero or Infospheres, we prefer to add new services and maintain them within the Web Java server addressing space.

Such organization facilitates management and offers natural, Web-browser based monitoring, publication and distribution support for the Web software.

Figs 2 and 3 illustrate our early experiments with Jigsaw where we constructed a chat collaboratory as Jigsaw resource (Fig 2) and we formed a token ring by connecting a set of Jigsaw resources viewed as WebFlow modules using Infospheres portlets. Later on, we switched to the Jeeves model since the servlet API is likely to become a standard as given by a core Java package `java.servlet`. We intend to continue the exploration of Jigsaw and other promising public domain Java systems and we tentatively base the WebFlow/WebVM prototype development on the Jeeves server architecture.

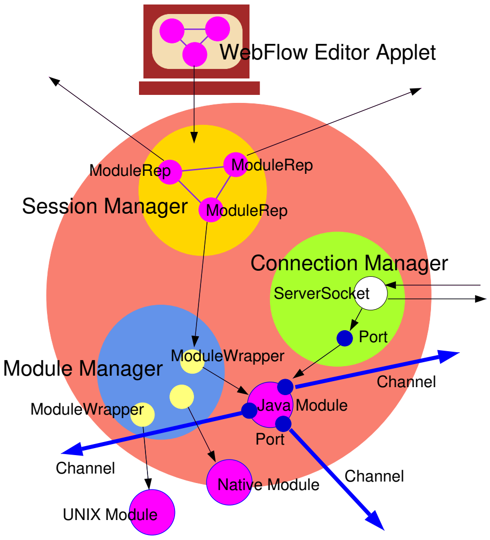


Figure 4: Initial design of the WebFlow management layer, implemented as a set of Jeeves servlets and including: Session Manager, Module Manager and Connection Manager.

5 Tier-2 WebFlow/WebVM Prototype

5.1 Overview Our prototype WebVM is given by a mesh of Jeeves servers, running servlets that manage and coordinate distributed computation. Atomic encapsulation units of WebVM computation are called *modules* and they communicate by sending objects along *channels* attached to module *ports*. Unlike management servlets which are usually persistent and application independent, modules are more transient and can be dynamically created, connected, scheduled, run, relocated and destroyed by servlets. WebFlow is a particular programming paradigm implemented over WebVM and given by a dataflow programming model (other models under experimentation include data parallel, collaboratory, and televirtual paradigms). WebFlow application is given by a computational graph, visually edited by end-users using Java applets.

Modules are written by module developers, people who have only limited knowledge of the system. on which the modules will run. They not need concern themselves with issues such as:

- allocating and running the modules on various machines
- creating connections among the modules
- sending and receiving data across these connections
- running several modules concurrently on one machine

The WebFlow system hides these management and coordination functions from the developers, allowing them to concentrate on the modules being developed.

WebFlow management is currently implemented in terms of the following three servlets: Session Manager, Module Manager, and Connection Manager. These servlets are URL addressable and can offer dynamic information about their services and current state. Each of them can also communicate with each other through sockets as discussed in the next section.

Fig 4 illustrates the three base servlets employed in setting up and managing WebFlow operation. Session Manager receives graph specification from the editor applet, creates an image of the whole compute-web using module proxy objects called ModuleRepresentation, decides on the compute-web decomposition strategy, and notifies Module Manager about local modules to be instantiated.

Module Manager starts and maintains ModuleWrapper threads than run Modules. Each module, when created, notifies ConnectionManager about the connectivity required by this module Ports, and waits for the connections to be established.

WebFlow channels connecting two module Ports are formed dynamically by the corresponding ConnectionManagers: Sockets returned by their 'accept' and 'connect' calls are passed to the appropriate ports. After all ports of a module receive their requested sockets, the module notifies the Module Manager and is ready to participate in the dataflow operations.

5.2 WebFlow requirements The requirements placed on WebFlow stem from the discussion above. Namely, WebFlow shall:

- allow modules to be run on demand
- support communication between the modules
- provide facilities for the user to create and destroy an application, where an application is a set of interconnected modules.

To support the requirements placed on the system, the following components have been created:

- Module Manager, in charge of running modules on demand
- Connection Manager, in charge of creating connections between the modules
- Session Manager, in charge of executing all the actions the user performs on the front end.

In the following section, we describe each of these management entities in more detail.

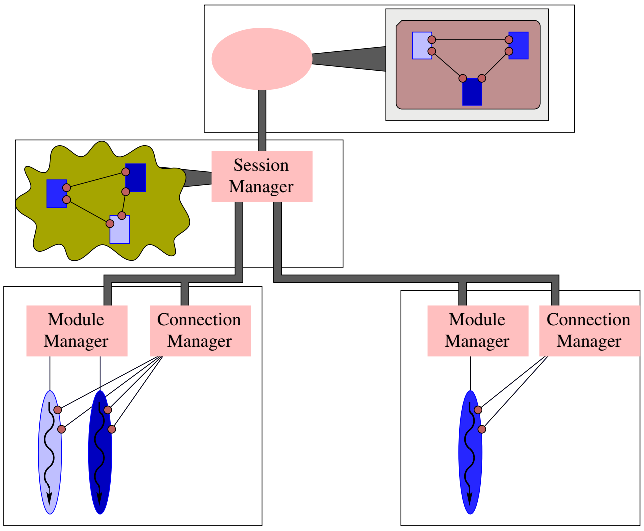


Figure 5: Overview of WebFlow Prototype Design

5.3 WebFlow management

Module Manager The Module Manager is the simplest of the three system components. It is in charge of running modules on demand. A user/editor request to create a module is sent to the Module Manager residing on the particular machine on which the module should be run. The Module Manager creates a separate thread for the module (thus enabling concurrent execution of multiple modules), and loads the module code, making the module ready for execution.

A request for running (destroying) a module triggers a special method called *run* (*destroy*). These methods were written by the module developers.

An important observation is that the Module Manager has no notion of a session built into it. It can support any number of modules, and requests coming from any number of Session Managers.

Connection Manager The Connection Manager is in charge of establishing connections between modules. To be precise, it establishes connections between individual ports, regardless of the module on which they reside, and regardless of the machine on which the module is run.

As each module is initialized, its ports register with the Connection Manager. This enables the Connection Manager to establish connections between registered ports as illustrated in Fig. 6.

To connect port 1 and port 2 in Fig. 6, a *connect* request is received by the first Connection Manager in step 1. In step 2, an *establish* request is sent to the second Connection Manager, which then, in step 3, sends an OK message back to the first Connection Manager to acknowledge the *establish* request. The second Connection Manager proceeds to send a Connection back to the first Connection Manager which receives the connection and passes it on to the port. Finally, in step 5, the first Connection Manager replies that the operation has succeeded. If an error occurs in any stage of the protocol, then instead of OK messages, error messages will be sent back, thus aborting the protocol, and notifying the caller that the connection failed.

The figure shows the more general case in which the two ports reside in separate Connection Managers. Of course, the two ports may be registered at the same Connection Manager, in which case the whole connection procedure is simplified, and steps two and three are not needed. As with the Module Manager, the Connection Manager has no notion of a session built into it. It can support any number of ports, and requests coming from any number of Session Managers.

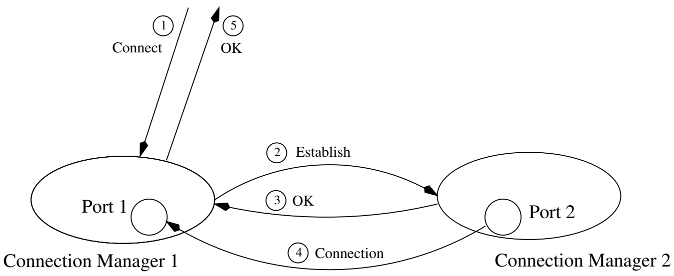


Figure 6: Steps involved in making the connection between the two ports

Session Manager The Session Manager is the part of the system in charge of accepting user commands from the front end, and executing them by sending requests to the rest of the system. These requests include: creating a new module, connecting two ports, running the application, and destroying the application.

Both the Session Manager and the front end store a representation of the application that the user is building. The difference between the two is that the Session Manager needs to worry about the machines on which each of the modules has been started, while the front end worries about the position of the representation of the module on the screen.

In the WebFlow prototype, the Session Manager can only work with one user at a time. In other words, there is only one session active at any one point in time (we are currently exploring JSDA support for WebFlow to provide multi-user collaborative editing capabilities).

5.4 Internal communication in WebFlow As illustrated in Fig. 7, the WebFlow prototype supports two types of communication:

- via URL
- via socket connections

In the figure, the client can either be front end, or the Session Manager, while the servlet can be any of the three servlets that exist in the system.

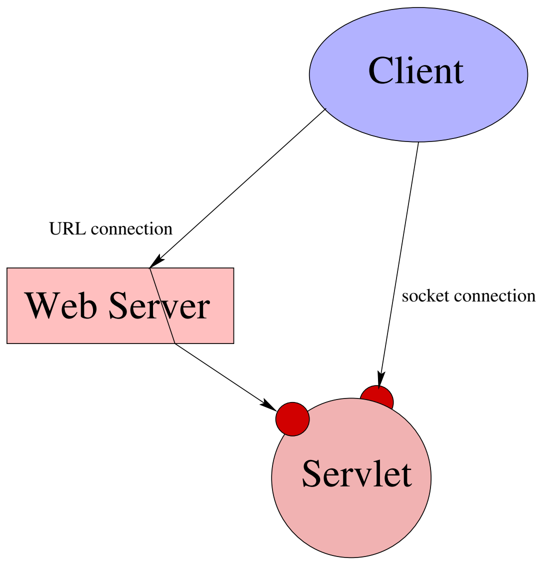


Figure 7: Servers in the WebFlow system are accessible through both URL and socket connections.

All the URLs point to the web server. The web server analyzes the URL, and forwards the request to the servlet denoted in the URL. Socket connections are received directly by the servlet.

The former - via URL - is used when a component's socket address is unknown. This feature allows the whole system to be accessed over the web. However, the current implementation of the URL addressing scheme does not provide a convenient way to send whole objects as parameters. On the other hand, the socket connection scheme provides for a very natural way of sending any object, provided it knows how to serialize itself, over the socket. This is very useful, as all the requests and replies can easily be expressed as objects whose internal state holds the type and parameters of the request or reply.

At the time being, each of the three servlets in the WebFlow system listens to both the URL and socket connections at all times. Internal requests and replies for creation, running, and destruction of modules, as well as connecting ports all go through the socket connection, whereas the URL communication is being used to provide the socket address of the server and to perform system-wide operations, such as give usage statistics, reset the system, add new resources to the system, etc.

It is conceivable that the HTTP protocol will evolve so that the whole WebFlow communication could be eventually handled uniformly in the URL addressing mode. For the time being, we will support both URL and socket based addressing modes and we will monitor, participate in and respond to W3C efforts aimed at dynamic and object-oriented extensions of the HTTP protocol.

5.5 Module and Port identification in WebFlow As soon as a module is created, it is assigned a unique identifier. This identifier is present with all the requests associated with the module, i.e. the module's running and destruction (recall that creation also creates the identifier). Module identifiers are necessary because of the following reasons:

- they provide an easy way of identifying the target of module operations
- they enable multiple instances of the same module to be run on the same machine, each of the instances having a separate identifier

Each port also has an identifier, but they are less general than the module identifiers. Since ports can never exist outside of a module, it suffices to assign unique identifiers to ports on one module. The current implementation is a bit more general, however, since it assigns identifier to ports per one Module Manager/Connection Manager combination.

5.6 I/O modules in WebFlow The previous discussion took for granted the input and output modules in an application. However, current web browser restrictions make input and output modules a non-trivial task.

Since the front end can be invoked from an arbitrary machine connected to the web, the input and output modules should be able to receive their input and send their output to the same, arbitrarily chosen, machine. The only way of doing so in the current state of affairs is to provide applets that will be able to receive user inputs, and show the application's outputs.

Therefore, the input/output modules are made of two parts: a WebFlow part - that works under the WebFlow model, and an applet part - that provides I/O capabilities, as illustrated in figure 8. Upon initialization, the I/O modules inform the system that they require an applet to be spawned for them. That request is forwarded all the way to the system's front end, which has the capability to open a new frame on the screen, and load an HTML page in it. That HTML page can contain an invocation of the I/O module's applet.

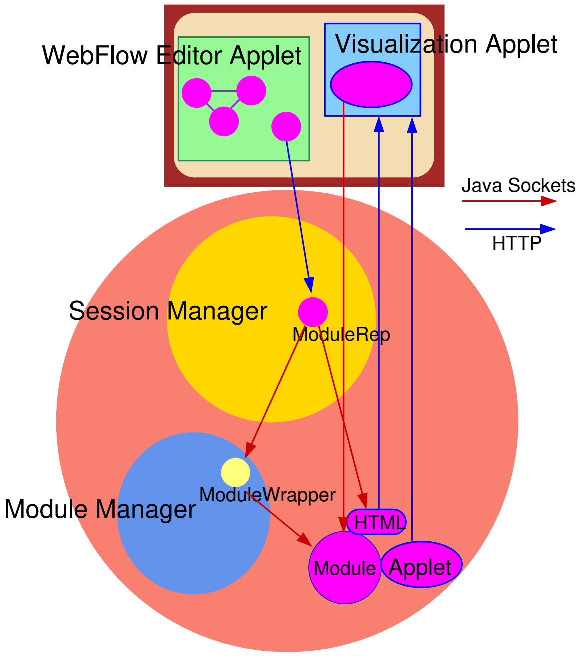


Figure 8: I/O Modules in WebFlow

The front end receives the HTML pages by making separate requests to the Session Manager. In the long run, the responsibility of creating and serving these HTML pages will be placed in a separate manager - the Viewer Manager, a topic further discussed in the following sections.

5.7 WebFlow API WebFlow offers a well-defined API for module developers that hides the communication details in terms of port and module abstractions. We include here for illustrative purposes a few samples of WebFlow programming at the module developer level.

Ports It is fairly easy to create and add new ports in the module implementation. Any new port type has to be derived from abstract Port class. The new port type has to only override the send and receive methods of the Port class for data transfer. The Port class constructor automatically registers the port with the Connection Manager. When the module terminates, the port deregisters itself from the Connection Manager.

An example port is shown below. The port is an Integer port which sends and receives Integer objects.

```
public class IntPort extends Port {
   DataInputStream is;
   DataOutputStream os;
   Integer data;
   boolean bool;

   public void send(Object num)         {
      data= (Integer)num;
      if(getSocket()!=null) {
         try {
            os= new DataOutputStream
            (getSocket().getOutputStream());
            os.writeInt(data.intValue());
         }  catch( IOException e){}
      }
   }

   public Object receive() {
      if(getSocket()!=null) {
         try {
            is = new DataInputStream
            (getSocket().getInputStream());
            data= new Integer(is.readInt());
         } catch (IOException e){}
         return(data);
      }
      else return(null);
   }
}
```

Ports can be both synchronous as well as asynchronous depending upon the way they check for data. Asynchronous ports remain dormant and wake up whenever data is available for receiving or sending. Synchronous ports keep on polling for data, so the user is flexible to use synchronous or asynchronous ports depending upon the application.

Modules

Modules basically consist of three main methods:

- initialize
- run
- destroy

Code for a basic adder module is shown below. This module receives numbers from two other modules and sends the result to a third module, as shown in Fig 9..

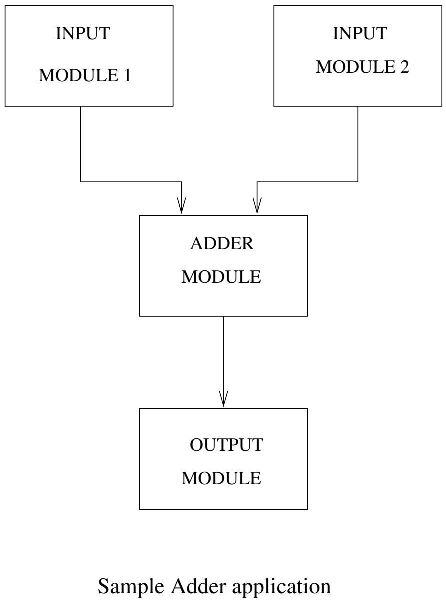


Figure 9: Sample Adder Application

The *initialize* method initializes the module by registering its ports with the Connection Manager, creating a MetaModule object containing the module id and port id's and then passing on the MetaModule object to the Module Manager. Essentially, all ports are declared and instantiated in this method. For example,

```
public MetaModule initialize() {
      //set the Metamodule
   MetaModule mm = new MetaModule("mm");
      //declare the ports
   port1= new IntPort();  //input port1
   mm.putPortID(port1.getPortID());
   port2= new IntPort();  //input port2
   mm.putPortID(port2.getPortID());
   port3= new IntPort();  //output port
   mm.putPortID(port3.getPortID());
   return(mm);
}
```

Viewer module specifies additionally an HTML string to be passed to the front-end and used there to fire a suitable viewing frame. In the initialize method, the MetaModule object holds this string and passes it on to the ModuleManager. The HTML syntax may contain code to display images, run other applets, etc. An example of the initialize method of the viewer module is given below.

```
public MetaModule initialize() {
      // Set the MetaModule  MetaModule
   mm = new MetaModule("mm");
   int i;
      //declare the port
   imgport=new ImgPort();
   mm.putPortID(imgport.getPortID());
      //data reqd for this particular module
   try{
      InetAddress local =
        java.net.InetAddress.getLocalHost();
      hostName = local.getHostName();

      listener = new ServerSocket(0);
        // open new server socket
      portNumber = listener.getLocalPort();
   }
   catch (UnknownHostException e) {
      System.out.println(e);
   }
   catch(IOException e ) {
      System.out.println(e);
   }
     // create the HTML String object
   String htmlString = new
             String(...HTML code...);
     //Store the object in the MetaModule
   mm.setHTML(htmlString);
     //return the Metamodule
   return(mm);
 }
```

The *run* method describes the behavior of the module. Upon receiving the run request from the ModuleManager, the module executes the *run* method in which the module may receive, send or process data. It is here that the module can interact with various other modules by data transfer.

```
public void run() {
   while(true)  {
         //receive values
      num1=port1.recieve().intValue();
      num2=port1.recieve().intValue();
      num3=num1 + num2;
         //send result
      port2.send(new Integer(num3));
   }
}
```

The *destroy* method terminates a running module. All ports are deregistered and the module stops executing. All socket connections of the ports are closed.

```
public void destroy(){   //terminate
   port1.destroy();      //destroy all ports
   port2.destroy();
   port3.destroy();               }
```

5.8 Next Steps The WebFlow prototype served its role as a proof that such a system can be built but it also showed that several new servers are needed to provide the full functionality. One of them — the Viewer Manager — was already mentioned above. At least the following new servlets will be added to the WebFlow system:

- Viewer Manager
- WebVM Server Manager
- Resource Manager
- Communication Manager

The Viewer Manager will be in charge of providing HTML pages that include I/O module's applet. One Viewer Manager will reside with every Module Manager, since any Module Manager may have I/O modules.

The WebVM Server Manager will be responsible for managing the servers in the system. It will be capable of adding and deleting servers from the system, as well as responding to queries about active servers. Unlike Module Managers and Connection Managers, Server Managers will be scarce in the system.

The Resource Manager will provide a list of resources, or modules, that can be found in the system. There are two possible ways of implementing resource management functions. One is by assigning a dedicated Resource Manager to each host, and the second is via a more collective Resource Manager, responsible for a group of hosts. In the first case, the Resource Manager could be grouped with the Module Manager, and in the second, it could be grouped with the Server Manager.

The Communication Manager will multiplex all the communications between ports registered on a given WebVM node. In the WebFlow prototype, each port has its own socket through which it communicates with the remote port, thus not only wasting system resources (one extra socket per port), but also having to deal with the low level details of sending and receiving messages (although it has the distinct advantage of having the sockets themselves take care of message buffering).

The Communication Manager will provide facilities for sending, receiving, and buffering messages. Its natural place is together with the Connection Manager, since these two servlets actually represent only two stages in the overall communication process. Future WebFlow implementations will probably have just one Connection and Communication Manager, instead of two separate ones.

6 Tier-1 Visual Graph Editor

Since the idea of WebFlow is to create and maintain a domain of world-wide reusable computational modules, the natural place for accessing and maintaining such a domain is the Web itself. Therefore we are faced with the existing browsers such as Netscape or Internet Explorer as a basis for the WebFlow Graphical User Interface. The security restrictions imposed by these browsers, implementation differences due to the ongoing corporate competition, as well as the recent developments in the Network Computer domain all point towards a design solution of a light weighted front end, that will be accessible trough any browser (including new consumer electronic front-ends) and a solid back end given by a personal Java Web server, hooked to a WebVM network, which will implement the most of the functionality of the system.

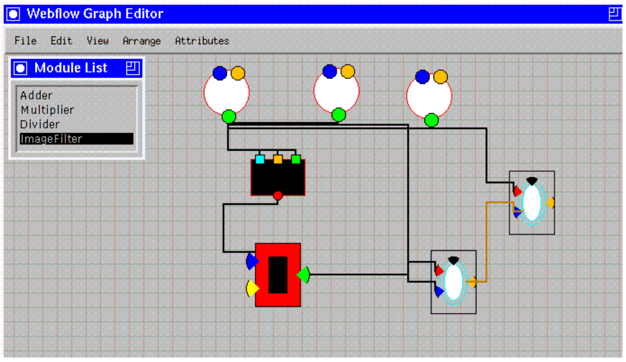


Figure 10: Initial WebFlow front-end, based on extended GEF (Graph Editing Framework) from UCI. Modules are selected from the palette in the click-and-drag style. Compute-webs are constructed interactively in the click-click-to-connect model. Individual modules can be given user-programmable visual appearance. In the next step, vector graphics drawing tools will be provided for interactive authoring of module icons.

The front end is designed as a tool for visual authoring of computational dataflow graphs that integrate the existing public domain software modules. It is based on highly intuitive visual icons and click-and-drag design metaphors which hide the inherent complexity of the WebFlow system.

In the current implementation of the front end we used the UCI's Graph Editing Framework (GEF) [Robb96] as a basis to develop the front end of the WebFlow system. GEF supports the basic graph editing mechanisms and it is naturally extensible. This framework is well structured with cleanly decoupled layers, which makes possible to concentrate on the application specific details that concern the WebFlow front end. Figure 10 shows a snapshot of the current editor in action.

The front end is implemented as an applet, it resides in the top level layer of the system, and it creates and maintains a connection with the Session Manager in the back end of the system.

The user creates a computational graph from modules as building blocks, by selecting the corresponding icons from a list of available modules in the system and inserting them into the graph. Multiple instances of a specific module can be created and their internal state and their connections are completely independent.

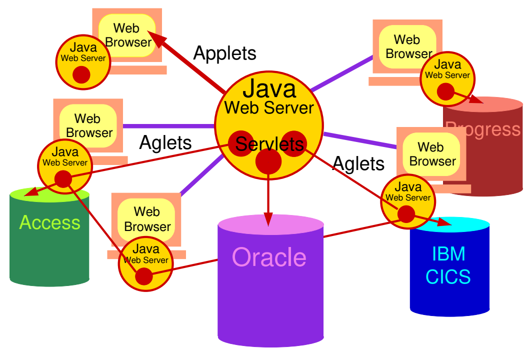


Figure 11: Example of a distributed heterogeneous database environment, managed by the WebVM layer via JDBC interfaces, and custom editable via WebFlow visual graph editing tools. Intelligent agents such as IBM aglets are used to disseminate or search for the information, stored/retrieved by servlets and visualized by applets.

After the modules are inserted as nodes in the graph, the applet requests its initialization from the back end. After the initialization is done the back end replies to the applet, bringing information about the interface of the selected module. The applet builds and stores the representation of the graph, keeping information just about its visual representation. The information about the actual modules and their mapping on real machines are stored by the back end.

In the same fashion as the modules, the connections between modules are created. Connecting two modules means connecting a port from one module to a port of the other, by means of simple clicking and dragging.

Individual modules and/or connections can be removed from the graph, which results in deleting them from the structure maintained by the applet itself and in killing the initialized instances of the corresponding modules in the real system as well as breaking the real connections between initialized modules.

After the computational graph is created it can be executed as well. The results are monitored trough the input/output modules that are inserted in the graph. The execution of a computational graph can generate variety of feedback patterns, ranging from just producing final results from a complex computation, to periodic performance visualization and system monitoring modes, to real-time interactive display modes. Current WebFlow editor is restricted to single-user 2D graphics operations but we are also initiating activities on bringing the front-end to the next level of interactivity. This includes integrating WebFlow with JSDA to support collaboratory editing and with VRML2 to support televirtual authoring paradigm.

7 Tier-3 Legacy Layer

In parallel with the core WebFlow development work described so far, we are also starting activities on building domain-specific tier-3 module libraries, including WebFlow wrappers to existing codes and legacy systems. ModuleWrapper discussed in Section 5 can wrap any computation, including pure Java, native libraries or external UNIX or NT processes.

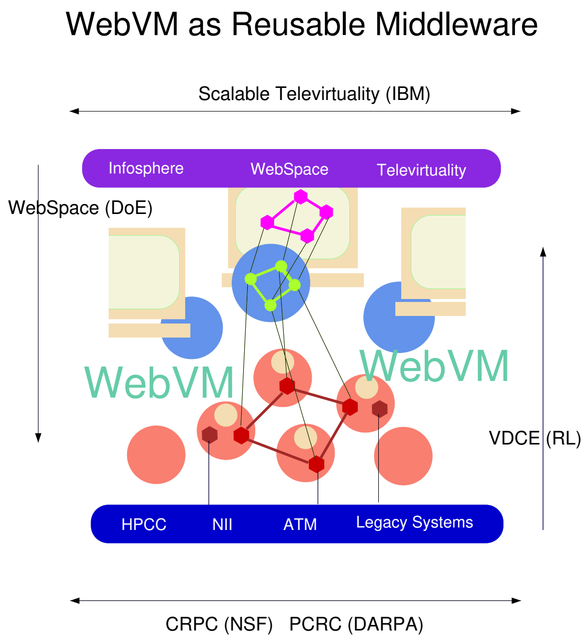


Figure 12: WebVM as a reusable middleware, tested in a set of research projects at Syracuse University such as WebSpace, VDCE and Televirtuality, focused on various front-end metaphors in tier-1 and/or various computational paradigms in tier-3.

In the pure Java sector, we are developing control, monitoring and coordination support for the base WebVM/WebFlow operations. Native libraries with C-coded optimized primitives offer a natural extension for media processing and high performance computing . In the external processing sector, we are experimenting with JDBC drivers for Oracle and mSQL, with JDBC/ODBC drivers for PC databases such as Access or SQL Server and we are developing a WebVM based distributed database layer (see Fig. 11), with intelligent agent (such as IBM aglets) based connectivity and visual WebFlow support for designing high level information retrieval and data mining strategies.

8 WebFlow/WebVM Applications

We view WebVM as a reusable middleware and we intend to test it in a set of Web based distributed applications under development. These efforts, partially supported by Department of Energy, Rome Laboratory and IBM Watson, allow us to test various aspects of the WebVM architecture as illustrated in Fig. 12. In two 'depth' projects, WebSpace and VDCE, we are probing selected tier-1 and tier-3 aspects of WebVM, respectively. In the 'breadth' area, focused on system scalability we are initiating collaboration with IBM Watson in the area of Televirtuality and we are seeking federal funds to address World-Wide Virtual Machine architecture [HPDC96] [SC96]

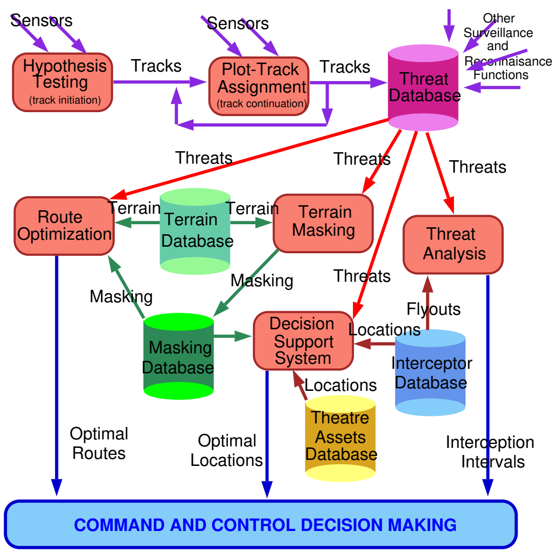


Figure 13: WebFlow/VDCE application for Command and Control

8.1 Command and Control

In the VDCE project [VDCE96], we are analyzing C3I functions recently published by the RL C3I Parallel Benchmarking Project and we are developing a library of 3CI modules that would support interactive composition of Battle Management C3I systems such shown as in Fig. 13 using the visual graph editing tools. More generally, VDCE addresses complementary aspects of Web based distributed computing and it offers a natural connectivity between the pure Java based WebFlow model and the ATM based HPDC environments.

8.2 Telemedicine

In the CareWeb project [CareWeb96], conducted jointly with Syracuse University College of Nursing, SUNY Health Science Center and Syracuse City School District, we are developing a collaboratory telemedicine system for school nursing, based on the 'bridge' topology [Bridge96]. Fig. 14 illustrates a CareWeb bridge under development, connecting 'points of need' (parents, nurses) with 'points of care' (nurse practitioners, pediatricians) via an intelligent Web based switchboard. Individual bridge services are managed as WebVM nodes and connected, integrated and customized for individual healthcare provider needs using the WebFlow visual authoring tools.

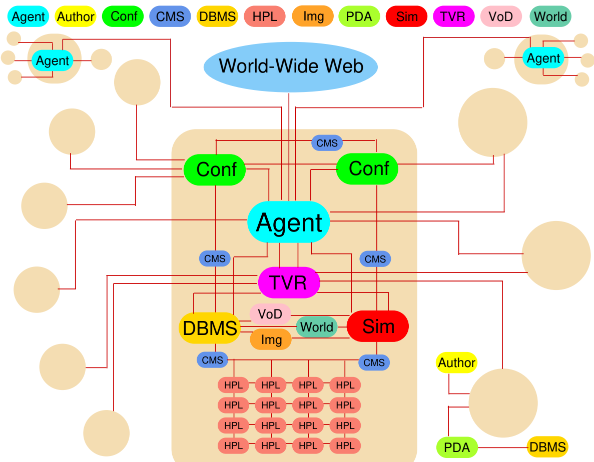


Figure 14: WebFlow based telemedicine bridge authoring toolkit

8.3 Televirtuality

In a joint project with IBM Watson [TVR96], we are analyzing scalability issues of WebVM architecture in the context of televirtual, i.e. 3D multi-user collaboratory environments on the Internet.

We use Java based Liquid Reality VRML2 browsers for the interactive front-ends and we start building 3D worlds that would provide experimentation platform for the scalability research.

We selected urban architectural domain for world building due to its natural modularity and we develop two specific worlds: Virtual SU Campus based on CAD data by the SU Department of Architecture, and Virtual Shopping Mall jointly with IBM Watson.

Our initial Mall architecture is drafted in Fig. 15. Individual stores, floors and towers are powered by WebVM servers, managed by the Mall tenants and offering interactive shopping services. WebFlow authoring tools will be used for specifying connectivity between architectural, commerce and human/avatar components of such a complex world.

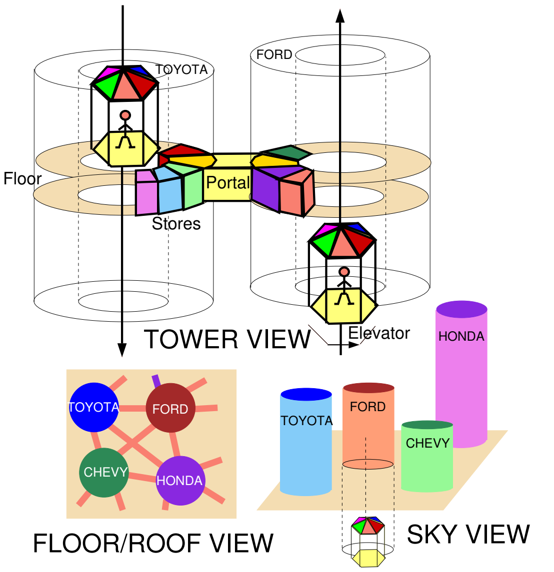


Figure 15: Example of a Televirtuality application with non trivial compute-web topology: Virtual Shopping Mall

We are also working on building system level WebFlow tools for performance visualization and interactive debugging, based on the TVR metaphor. In this world, modules are represented as rooms, ports as doors, channels as halls connecting rooms etc. New object arriving at an input port would result in the corresponding door opening and an avatar-messenger entering the room with a new chunk of data to be taken over by other avatars-managers for further handling.

9 References

[Bridge96] David Warner, D. and Balch, D., "Medicine Meets VR: BRIDGE", http://www.telemed.med.ecu.edu/bridge/bridge_1.htm

[Browne92] J.C. Browne, "CODE – The Computationally Oriented Display Environment", University of Texas at Austin, 1992
`http://www.cs.utexas.edu/users/code`

[CareWeb96] SU College of Nursing, NPAC, Syracuse School District, SUNY HSC, "CareWeb – a Web based community oriented healthcare communications system",
http://www.npac.syr.edu/projects/careweb

[ChaRi96] Mani Chandy, Adam Rifkin, "The Caltech Infospheres Project"
`http://www.cs.caltech.edu/~adam/CALTECH/infospheres.html`

[Dong94] J. Dongarra "HeNCE: Heterogeneous Network Computing Environment"
`http://www.cs.utk.edu/netsolve/`

[Haba96] NCSA's Habanero Collaborative Tools Library
`http://www.ncsa.uiuc.edu/SDG/Software/Habanero/ToolsLibrary.html`

[HPDC96] G. Fox and W. Furmanski, "Towards Web/Java based High Performance Distributed Computing – an Evolving Virtual Machine"
http:/www.npac.syr.edu/projects/webspace/doc/hpdc5/talk

[Jigs96] Jigsaw HTTP Server, World-Wide-Web Consortium,"
`http://www.w3.org/pub/WWW/Jigsaw/`

[Robb96] Jason Robbins, "GEF: Graph Editing Framework","
`http://www.ics.uci.edu/~jrobbins/GraphEditingFramework.html`

[SC96] W. Furmanski and W. Fox, "HyperWorld — Design and Prorotype Components"
http:/www.npac.syr.edu/projects/webspace/doc/sc96/talk

[TVR96] D. Dias, W. Furmanski, V. Mehra "Scalability of Televirtuality Servers"

[VDCE96] S. Hariri, G. Fox, W. Furmanski and S. Warzala, "Virtual Distributed Computing Environemnt" http:/merlin.cat.syr.edu/projects/vm

[WebHPCC96] Fox, G.C. and Furmanski, W., "Web based HPCC at NPAC", http://www.npac.syr.edu/projects/webspace/webbasedhpcc.html